# UNIVERSIDAD AUTÓNOMA DE MADRID

## ESCUELA POLITÉCNICA SUPERIOR




**Grado en Ingeniería Informática (GII)**

# TRABAJO FIN DE GRADO

## ANÁLISIS DE ALGORITMO DE REGRESIÓN MULTIOBJETIVO


**Ricardo Sánchez Berral**

**Tutor: Gonzalo Martínez Muñoz**


**JUNIO 2017**

# ANÁLISIS DE ALGORITMO DE REGRESIÓN MULTIOBJETIVO

**AUTOR: Ricardo Sánchez Berral**

**TUTOR: Gonzalo Martínez Muñoz**

**Escuela Politécnica Superior**

**Universidad Autónoma de Madrid**

**Junio de 2017**

# Resumen

Este Trabajo de Fin de Grado define qué es la regresión multi-objetivo, sus posibles ventajas e inconvenientes, explica cuál es el estado del arte sobre este tipo de regresión y detalla una implementación de regresión multi-objetivo sobre árboles CART y bosques aleatorios de árboles de decisión, utilizándola para explorar las diferencias con sus versiones mono-objetivo.



# Abstract

This Bachelor Thesis defines what multi-objective regression is, its possible advantages and pitfalls, it explains what the state of the art is about this type of regression, and it describes in detail a multi-objective implementation of regression algorithms such as CART trees and decision tree random forests, using said implementation to explore the differences between them and their mono-objective counterparts.



# Palabras clave

regresión, multi-objetivo, mono-objetivo, CART, árbol, bosque aleatorio



# Keywords

regression, multi-objective, mono-objective, CART, tree, random forest

*Agradecimientos*

***Quiero agradecer a mis padres y a mi hermana por no perder la esperanza en mí, y a José Luis por darme fuerzas para continuar.***

# ÍNDICE DE CONTENIDOS

# ÍNDICE DE FIGURAS



# ÍNDICE DE TABLAS

# 1 Introducción

## 1.1 Motivación

El campo del aprendizaje automático ha atraído cada vez más interés durante los últimos 4 años [1]. Problemas que antes parecían imposibles de resolver están cada día más al alcance del público general, no sólo gracias a los avances en *hardware*, sino también a los progresos en materia de algoritmos de aprendizaje automático. La mejora en la eficiencia y la fiabilidad en algoritmos de aprendizaje automático es por tanto imperativa, sobre todo cuando hablamos de problemas en tiempo real, como pueden ser la conducción autónoma o una simple aplicación móvil que superpone una máscara en las caras humanas presentes en un *feed* de vídeo.

En los ejemplos mencionados, el número de variables que se necesita predecir es mayor a uno. En lugar de predecir cada variable de forma independiente, una alternativa atractiva sería predecir todas las variables objetivo con un solo modelo; si hablamos de variables continuas, llamamos a este tipo de predicción **regresión multi-objetivo**. En este TFG exploramos los principios de la regresión multi-objetivo.

## 1.2 Objetivos

El principal objetivo de este trabajo de fin de grado es establecer una primera toma de contacto con los algoritmos de regresión multi-objetivo, re-implementar modelos conocidos como el árbol CART y el bosque aleatorio para que puedan realizar regresión multi-objetivo, y comparar la exactitud y la eficiencia programática de estas re-implementaciones con sus versiones mono-objetivo.

## 1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- *Introducción*: damos una idea de las posibles aplicaciones de los modelos de regresión multi-objetivo, y describimos nuestros objetivos y la organización de la memoria.
- *Estado del arte*: detallamos brevemente los fundamentos de los árboles CART y bosques aleatorios, y aportamos un resumen del trabajo actual de estudio de sus versiones multi-objetivo.

- *Implementación de la solución*: aportamos una perspectiva de la biblioteca estadística con las implementaciones originales de los modelos estudiados, especificamos el desarrollo realizado para re-implementar esos modelos, aportando una descripción detallada del código, y listamos las pruebas realizadas para garantizar la corrección del nuevo código.
- *Pruebas experimentales*: experimentamos con nuestras re-implementaciones multi-objetivo del árbol CART y del bosque aleatorio, utilizando 2 modelos de datos, uno sintético y el otro generado a partir de simulaciones cosmológicas.
- *Conclusiones y trabajo futuro*: recapitulamos las observaciones obtenidas a través de las pruebas experimentales, y pormenorizamos las conclusiones obtenidas. Además, ofrecemos posibles vías para trabajo futuro.
- *Referencias*: listado de la documentación, artículos, páginas *web* y demás material utilizado o consultado para la realización de esta memoria.
- *Glosario*: diccionario de los tecnicismos y términos específicos utilizados a lo largo de esta memoria.
- *Anexos*: todo el material original relevante para este Trabajo de Fin de Grado, pero cuya presencia en el cuerpo de la memoria no es imprescindible.

# 2 Estado del arte

## 2.1 Árboles CART

Un árbol CART es un árbol de decisión binario construido de tal forma que, empezando con un solo nodo raíz que engloba a todos los nodos del conjunto de entrenamiento, se escoge un criterio para separar los nodos en dos clusters de tal forma que la "distancia" (una métrica elegida por el usuario) entre los nodos de cada cluster sea mínima.

El criterio de parada engloba varias condiciones que, en caso de cumplirse al menos una de ellas, terminan el entrenamiento del árbol [2]:

- Si un nodo se vuelve puro, es decir, los valores de la variable dependiente de las instancias del nodo son iguales.
- Si los valores de las variables independientes de las instancias del nodo son iguales.
- Si la profundidad del árbol alcanza una profundidad máxima, impuesta por el usuario.
- Si el número de instancias del nodo es menor que un tamaño nímino impuesto por el usuario, o si su corte da lugar a un nodo cuyo tamaño es menor.
- Si la mejora de impureza para el mejor corte es menor que la mejora de impureza mínima impuesta por el usuario.

Entre las razones por las cuales se considera al árbol CART como una de las herramientas más exitosas de final del siglo XX [3] se encuentran su flexibilidad a la hora de aplicarlo a problemas de clasificación o de regresión sin necesidad de asunciones formales sobre la estructura de los datos, la posibilidad de utilizarlo para conjuntos de datos de gran tamaño, su resistencia a la pérdida de datos, o la eficiencia con la que maneja las clases nominales. No sólo eso, sino que además su construcción ayuda a conocer mejor la naturaleza de los datos, ya que enfatiza las variables independientes que más influyen en la predicción, y ofrece en sus hojas un *clustering* de los datos más o menos homogéneo.

Sin embargo, su tasa de error es mayor que la de otros modelos de clasificación como el SVM o como conjuntos de clasificadores creados a través de métodos como *bagging* o *boosting* [4].

## 2.2 Bosques aleatorios

Un bosque aleatorio es un algoritmo de clasificación o regresión donde se construye un conjunto de árboles de decisión en cuya construcción, durante el entrenamiento de cada nodo, se elige una cantidad determinada de variables independientes al azar para entre ellas encontrar la mejor separación. Los árboles crecen hasta su máxima profundidad, sin tener en cuenta ningún otro criterio de parada [5].

Las ventajas de los bosques aleatorios incluyen, entre otras, una menor varianza gracias a la randomización, la adaptación automática al tamaño del conjunto de entrenamiento (es decir, no se sobre-entrena) y mejoras de eficiencia, ya que 100 árboles de un bosque se podrían entrenar hasta al mismo tiempo que 3 árboles CART individuales [3].

La tasa de error de un bosque depende de 2 factores [6]:

- La correlación entre dos árboles cualesquiera del bosque. A más correlación, más tasa de error del bosque.
- La tasa de error individual de cada árbol; disminuir el error para cada árbol conlleva disminuir el error para el bosque.

## 2.3 Árboles multi-objetivo

Un árbol multi-objetivo, o MORT, es un árbol de regresión capaz de predecir varias variables dependientes continuas de una vez. La principal diferencia con el árbol CART es que cada hoja contiene un vector con las predicciones para cada unas de las variables dependientes [7]. Otra diferencia con los árboles CART es que, tal y como lo definen sus creadores, la métrica utilizada en los MORT para encontrar el mejor corte es la varianza intra-*cluster*, definida como:

$$N \cdot \sum_{t=!}^{T} Var[y_t]$$

Siendo $N$ el número de ejemplos en el *cluster*, $T$ el número de variables dependientes, y $Var[y_t]$ la varianza de la variable dependiente en el *cluster* [7].

# 3 Implementación de la solución

## 3.1 Código ya existente

Para implementar los árboles CART y bosques aleatorios multi-objetivo, hemos utilizado un proyecto con utilidades estadísticas ya existente, cuyo administrador principal es el Dr. Gonzalo Martínez.

Dada la complejidad de la biblioteca, evitamos hacer un análisis pormenorizado de todas las clases, y en su lugar, describimos el funcionamiento general para un ejemplo clásico de regresión utilizando un árbol CART.

Antes de nada, debemos explicar que, para permitir al usuario usar la biblioteca, se utiliza un lenguaje de dominio específico, que define una serie de funciones de interfaz relevantes para un ejercicio estadístico. Esas funciones se utilizan en *scripts*, que la biblioteca interpreta y ejecuta.

En la Figura 1 podemos ver el flujo de control de un *script* para la creación y evaluación de un árbol CART. La creación de un bosque aleatorio sería equivalente, con la salvedad de que se sustituiría la función ***ConstruyeCART*** por ***ConstruyeRandomForest***.

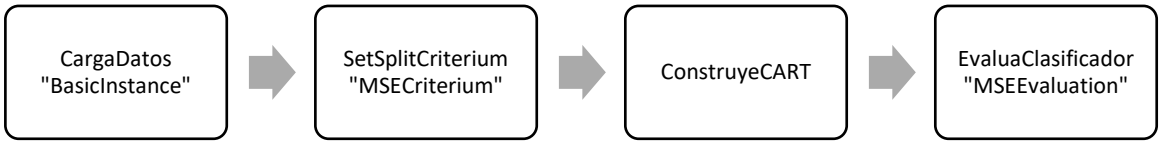


**Figura 1 Creación de un árbol CART**

Como se puede observar en la Figura 1, se utilizan como argumentos los nombres de las siguientes clases:

- *BasicInstance*: representa una fila de un fichero de datos, interpretando la última variable de la fila como la variable dependiente.
- *MSECriterium*: criterio para construir el clasificador, basado en el error cuadrático medio, y teniendo en cuenta la última variable de la instancia.
- *MSEEvaluation*: criterio para evaluar un clasificador, basado en el error cuadrático medio, y teniendo en cuenta la última variable de la instancia.

## 3.2 Nuevo código

El nuevo código se ha desarrollado en un entorno Ubuntu 17.04, utilizando las siguientes herramientas:

- **Visual Studio Code**
- **GitHub**: el código de este Trabajo de Fin de Grado está alojado en https://github.com/RicardoSBerral/moras
- **Python**, en su versión 2.7
- **Gnuplot**, en su versión 5.0

Para poder otorgar una perspectiva multi-objetivo a la implementación de árbol CART y de bosque aleatorio ya creadas, lo primero que hemos necesitado es implementar alternativas a las clases utilizadas como argumentos en la Figura 1:

- *MultiobjectiveInstance*: representa una fila de un fichero de datos, y permite al usuario elegir el número de variables dependientes.
- *MultipleMSECriterium*: criterio para construir el clasificador, basado en el error cuadrático medio, y teniendo en cuenta las variables dependientes elegidas al leer los datos.
- *MultipleMSEEvaluation*: criterio para evaluar un clasificador, basado en el error cuadrático medio, y teniendo en cuenta las variables dependientes elegidas al leer los datos.

Además, con el objetivo de realizar pruebas de la corrección del nuevo código, además de pruebas experimentales sobre los nuevos clasificadores multi-objetivo implementados, se han añadido las siguientes funciones de interfaz:

- *ChangeDependentColumn*: intercambia los valores de la variable dependiente por otra variable que el usuario ha elegido.
- *CopyDependentColumn*: copia los valores de la variable dependiente, de tal forma que se genere una nueva variable dependiente idéntica a la original.

Se han realizado adicionalmente pequeñas modificaciones; un análisis más detallado de esas modificaciones se encuentra en el Anexo A.

## 3.3 Integración, pruebas y resultados

Con el objetivo de comprobar la corrección del nuevo código, se han realizado las siguientes pruebas, cuya idea general es comprobar las versiones multi-objetivo con las versiones mono-objetivo, en tales circunstancias que el resultado esperado de la evaluación de los clasificadores sea el mismo.

**Árbol CART multi-objetivo con un solo objetivo**

Esta prueba consiste en comparar un árbol CART clásico con un árbol CART multi-objetivo, pero especificando 1 solo objetivo. Utilizamos la base de datos *WAVEFORM* (explicada en el apartado Metodología) para ejecutar la prueba. Efectivamente, cuando ejecutamos los comandos:

- `./nucleo/cons script_reg_tree.mKo`
- `./nucleo/cons script_reg_mtree.mKo`

Obtenemos una evaluación equivalente en ambos casos, como se puede comprobar en la Tabla 1.

**Tabla 1 Evaluación de árbol CART multi-objetivo con 1 objetivo**

| **Error evaluado en `./nucleo/cons script_reg_tree.mKo`** | **Resultados de `./nucleo/cons script_reg_mtree.mKo`** |
|---|---|
| `6: EvaluaClasificador(cart, dts, "MSEEvaluation")`<br>`0.6034`<br>`7: EvaluaClasificador(cart, dtr, "MSEEvaluation")`<br>`0` | `6: EvaluaClasificador(mcart, mdts, "MSEEvaluation", 21)`<br>`0.6034`<br>`7: EvaluaClasificador(mcart, mdtr, "MSEEvaluation", 21)`<br>`0` |

**Bosque aleatorio multi-objetivo con un solo objetivo**

Esta prueba es una versión utilizando bosques aleatorios de la prueba anterior. Consiste en comparar un bosque aleatorio clásico de 100 árboles con un árbol CART multi-objetivo, pero especificando 1 solo objetivo. Efectivamente, cuando ejecutamos los comandos:

- `./nucleo/cons script_reg_forest.mKo`
- `./nucleo/cons script_reg_mforest.mKo`

Obtenemos una evaluación equivalente en ambos casos, como se puede comprobar en la Tabla 2.

**Tabla 2 Evaluación de bosque aleatorio multi-objetivo con 1 objetivo**

| **Resultados de `./nucleo/cons script_reg_forest.mKo`** | **Resultados de `./nucleo/cons script_reg_mforest.mKo`** |
|---|---|
| `6: EvaluaClasificador(rf, dts, "MSEEvaluation")`<br>`0.29010126`<br>`7: EvaluaClasificador(rf, dtr, "MSEEvaluation")`<br>`0.040246` | `6: EvaluaClasificador(mrf, mdts, "MSEEvaluation", 21)`<br>`0.29010126`<br>`7: EvaluaClasificador(mrf, mdtr, "MSEEvaluation", 21)`<br>`0.040246` |

## Probando la función de interfaz CopiaColumnaDependiente

En esta prueba, comprobamos la corrección del comando *CopiaColumnaDependiente*. Para ellos, a partir del conjunto de datos *WAVEFORM*, generamos un fichero con una columna duplicada utilizando las siguientes expresiones regulares:

- *Búsqueda*: `((?:[^ \n]* ){21})([^ ]*)\n`
- *Reemplazo*: `$1$2 $2\n`

Guardamos este nuevo conjunto de datos en los ficheros *wtr_duplicado.cre* y *wr_duplicado.cre*. A continuación, suponiendo un número de objetivos igual a 2, creamos dos árboles CART, uno sobre el conjunto original, al que se le ha aplicado la función *CopiaColumnaDependiente*, y otro sobre el conjunto que hemos duplicado manualmente. Cuando ejecutamos los comandos:

- `./nucleo/cons script_reg_mtree_duplicado.mKo`
- `./nucleo/cons script_reg_mtree_duplicadoManual.mKo`

Obtenemos el mismo árbol, tal y como se observa en la Tabla 3.

**Tabla 3 Árboles CART con 2 objetivos idénticos**

| **Resultados de `./nucleo/cons script_reg_mtree_duplicado.mKo`** | **Resultados de `./nucleo/cons script_reg_mtree_duplicadoManual.mKo`** |
|---|---|
| `9: EvaluaClasificador(mcart, mdts, "MultipleMSEEvaluation")`<br>`0.6034`<br>`10: EvaluaClasificador(mcart, mdtr, "MultipleMSEEvaluation")`<br>`0` | `6: EvaluaClasificador(mcart, mdts, "MultipleMSEEvaluation")`<br>`0.6034`<br>`7: EvaluaClasificador(mcart, mdtr, "MultipleMSEEvaluation")`<br>`0` |

## Multi-objetivo con dos objetivos idénticos

Una vez que hemos comprobado que el comando *CopiaColumnaDependiente* es correcto, lo utilizamos para comprobar que un árbol CART clásico y un árbol CART multi-objetivo con 2 objetivos idénticos dan la misma predicción. Para ello, utilizamos los comandos:

- `./nucleo/cons script_reg_tree.mKo`
- `./nucleo/cons script_reg_mtree_duplicado.mKo`

Efectivamente, obtenemos dos árboles equivalentes que dan la misma predicción, como podemos comprobar en la Tabla 4.

**Tabla 4 Árbol CART clásico y árbol CART multi-objetivo con 2 objetivos idénticos**

| **Resultados de `./nucleo/cons script_reg_tree.mKo`** | **Resultados de `./nucleo/cons script_reg_mtree_duplicado.mKo`** |
|---|---|
| `6: EvaluaClasificador(cart, dts, "MSEEvaluation")`<br>`0.6034`<br>`7: EvaluaClasificador(cart, dtr, "MSEEvaluation")`<br>`0` | `9: EvaluaClasificador(mcart, mdts, "MultipleMSEEvaluation")`<br>`0.6034`<br>`10: EvaluaClasificador(mcart, mdtr, "MultipleMSEEvaluation")`<br>`0` |

## Probando la función de interfaz CambiaColumnaDependiente

En esta prueba, comprobamos la corrección del comando *CambiaColumnaDependiente*. Para ellos, a partir del conjunto de datos WAVEFORM, generamos un fichero en el que las dos últimas columnas se intercambian, utilizando las siguientes expresiones regulares:

- *Búsqueda*: `((?:[^ \n]* ){20})([^ \n]*) ([^ ]*)\n`
- *Reemplazo*: `$1$3 $2\n`

Guardamos este nuevo conjunto de datos en el fichero *wr_cambiado.cre* y *wrt_cambiado.cre*. A continuación, creamos dos árboles CART y dos bosques aleatorios de 100 árboles, unos sobre el conjunto original, al que se le ha aplicado la función *CambiaColumnaDependiente*, y otros sobre el conjunto donde se han intercambiado manualmente las columnas. Cuando ejecutamos los comandos:

- `./nucleo/cons script_reg_tree.mKo`
- `./nucleo/cons script_reg_tree_cambiado.mKo`

y

- `./nucleo/cons script_reg_forest.mKo`
- `./nucleo/cons script_reg_forest_cambiado.mKo`

Podemos comprobar en la Tabla 5 y en la Tabla 6 que los clasificadores obtenidos son equivalentes.

**Tabla 5 Árboles CART con las dos últimas variables intercambiadas**

| **Resultados de `./nucleo/cons script_reg_tree.mKo`** | **Resultados de `./nucleo/cons script_reg_tree_cambiado.mKo`** |
|---|---|
| `6: EvaluaClasificador(cart, dts, "MSEEvaluation")`<br>`0.6034`<br>`7: EvaluaClasificador(cart, dtr, "MSEEvaluation")`<br>`0` | `9: EvaluaClasificador(cart, dts, "MSEEvaluation")`<br>`0.6034`<br>`10: EvaluaClasificador(cart, dtr, "MSEEvaluation")`<br>`0` |

**Tabla 6 Bosques aleatorios con las dos últimas variables intercambiadas**

| **Resultados de `./nucleo/cons script_reg_forest.mKo`** | **Resultados de `./nucleo/cons script_reg_forest_cambiado.mKo`** |
|---|---|
| `6: EvaluaClasificador(rf, dts, "MSEEvaluation")`<br>`0.29010126`<br>`7: EvaluaClasificador(rf, dtr, "MSEEvaluation")`<br>`0.040246` | `9: EvaluaClasificador(rf, dts, "MSEEvaluation")`<br>`0.29010126`<br>`10: EvaluaClasificador(rf, dtr, "MSEEvaluation")`<br>`0.040246` |

# 4 Pruebas experimentales

## 4.1 Metodología

Para todos los experimentos y conjuntos de datos de este apartado, se ha utilizado una evaluación utilizando validación cruzada con 10 partes. Para dividir los datos en los conjuntos de entrenamiento y evaluación, se ha utilizado la función de interfaz *CreaConjuntosTrainAleat*. Esta función crea subconjuntos de datos con la estructura descrita en la .

| **Conjunto de entrenamiento** | **Conjunto de evaluación** |
|---|---|
| data_train_cv0.cre | data_test_cv0.cre |
| … | … |
| data_train_cv9.cre | data_test_cv9.cre |

Para cualquier resultado numérico de las pruebas experimentales, entrenamos y evaluamos el modelo de datos con las 10 parejas de subconjuntos, y calculamos la media y la desviación típica de esos 10 resultados. Se utiliza la definición de desviación típica sin corrección de Bessel [8], es decir, $s = \sqrt{\frac{1}{n}\sum_{i=0}^{9}(x_i - \bar{x})^2}$.

Para el control de flujo de los experimentos, el entorno de ***scripting*** utilizado ha sido Python. Aunque en un principio se valoró utilizar Bash, se ha preferido Python por los siguientes motivos:

- En Bash no hay operaciones aritméticas nativas con números de cadena flotante, por lo que habríamos tenido que recurrir a herramientas externas como *awk* [9] o *bc* [10].
- Para tareas como el cálculo de la desviación típica, o la extracción de resultados de la salida del ejecutable de la biblioteca, es necesario la reutilización de código. Aunque los scripts Bash tienen la opción de definir funciones [11], el paso de argumentos y la recuperación del retorno es menos intuitiva que en Python.

La versión del entorno Python elegida es la 2.7, por ser la versión que se instala por defecto en la última versión del sistema operativo Ubuntu, la versión 17.04. De esta manera, maximizamos la compatibilidad de nuestros *scripts* en la mayoría de las *workstations*.

Para realizar los experimentos, se han utilizado 2 **conjuntos de datos**:

- *Base de datos WAVEFORM*: es un conjunto de datos sintético, construido a partir de distintas mezclas de señales triangulares, y cuya única variable dependiente es la última [12]. Tiene 22 variables. Aunque la última variable fue concebida como una variable discreta con 3 clases, para los propósitos de estos experimentos la hemos considerado continua.
- *Base de datos MUSIC:* es una base de datos de simulaciones cosmológicas de clusters de galaxias, extraídos de simulaciones como el MareNostrum Universe o la simulación MultiDark [13]. Tiene 30 variables y 7 variables objetivo.

## *4.2 Árboles CART y bosques, mono-objetivo y multi-objetivo*

Nuestra prueba prueba experimental consiste en medir el error cuadrático medio de los árboles CART y los bosques aleatorios con su nueva implementación multi-objetivo para cada variable de los 2 conjuntos de datos. Utilizando validación cruzada, también calculamos la desviación típica de las 10 medidas tomadas en los 10 cruces. Además:

- Para los árboles CART, medimos el número de nodos del árbol generado. En el caso del árbol multi-objetivo, sólo se genera un árbol, por lo que sólo calculamos un número de nodos para todas las variables dependientes.
- Para los bosques aleatorios, establecemos un número de árboles fijo de 10 y de 100.

**Tabla 7 Comparación del MSE utilizando conjunto de datos WAVEFORM**

| | | 1º objetivo | 2º objetivo | 3º objetivo | 4º objetivo | 5º objetivo |
|---|---|---|---|---|---|---|
| **ÁRBOL CART** | | | | | | |
| **Mono-objetivo** | **MSE ± SD** | 2.20E+0 ± 1.606E-1 | 2.13E+0 ± 1.172E-1 | 2.06E+0 ± 1.366E-1 | 2.15E+0 ± 1.025E-1 | 5.41E-1 ± 5.425E-2 |
| | **N.º nodos** | 8.85E+3 ± 1.420E+1 | 8.85E+3 ± 1.834E+1 | 8.84E+3 ± 1.172E+1 | 8.85E+3 ± 1.221E+1 | 9.92E+2 ± 2.343E+1 |
| **Multi-objetivo** | **MSE ± SD** | 2.18E+0 ± 1.713E-1 | 2.15E+0 ± 1.284E-1 | 2.08E+0 ± 1.016E-1 | 2.01E+0 ± 1.305E-1 | 5.69E-1 ± 5.063E-2 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | **N.º nodos** | 9.00E+3 ± 0.000E+3 | | | | |
| **BOSQUE ALEATORIO, CON 10 ÁRBOLES** | | | | | | |
| **Mono-objetivo** | **MSE ± SD** | 1.20E+0 ± 6.481E-2 | 1.18E+0 ± 5.500E-2 | 1.13E+0 ± 7.778E-2 | 1.13E+0 ± 8.107E-2 | 2.73E-1 ± 2.156E-2 |
| **Multi-objetivo** | **MSE ± SD** | 1.21E+0 ± 6.112E-2 | 1.14E+0 ± 4.950E-2 | 1.11E+0 ± 9.004E-2 | 1.14E+0 ± 7.289E-2 | 2.85E-1 ± 1.973E-2 |
| **BOSQUE ALEATORIO, CON 100 ÁRBOLES** | | | | | | |
| **Mono-objetivo** | **MSE ± SD** | 1.08E+0 ± 6.824E-2 | 1.06E+0 ± 5.003E-2 | 1.02E+0 ± 7.101E-2 | 1.03E+0 ± 6.485E-2 | 2.42E-1 ± 2.086E-2 |
| **Multi-objetivo** | **MSE ± SD** | 1.09E+0 ± 6.952E-2 | 1.06E+0 ± 4.739E-2 | 1.02E+0 ± 8.185E-2 | 1.04E+0 ± 6.860E-2 | 2.53E-1 ± 1.699E-2 |

**Tabla 8 Comparación del MSE utilizando conjunto de datos MUSIC**

| | | **1º objetivo** | **2º objetivo** | **3º objetivo** | **4º objetivo** | **5º objetivo** | **6º objetivo** | **7º objetivo** |
|---|---|---|---|---|---|---|---|---|
| **ÁRBOL CART** | | | | | | | | |
| **Mono-objetivo** | **MSE ± SD** | 1.66E-2 ± 1.168E-2 | 1.65E-4 ± 2.847E-5 | 5.94E-5 ± 1.635E-5 | 2.16E-2 ± 7.565E-3 | 1.11E-1 ± 1.456E-2 | 1.55E+2 ± 1.910E+2 | 7.02E-12 ± 3.368E-12 |
| | **N.º nodos** | 3.62E+3 ± 0.000E+3 | 3.62E+3 ± 8.000E-1 | 3.62E+3 ± 0.000E+3 | 3.62E+3 ± 0.000E+3 | 3.62E+3 ± 0.000E+3 | 3.62E+3 ± 0.000E+3 | 3.59E+3 ± 3.945E+0 |
| **Multi-objetivo** | **MSE ± SD** | 1.59E-1 ± 5.507E-2 | 2.52E-3 ± 8.027E-4 | 3.22E-4 ± 8.987E-5 | 1.21E-1 ± 1.948E-2 | 3.21E-1 ± 9.565E-2 | 2.19E+3 ± 8.087E+2 | 4.92E+0 ± 1.825E+0 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **N.º nodos** | 3.49E+3 ± 6.132E+0 | | | | | | |
| **BOSQUE ALEATORIO, CON 10 ÁRBOLES** | | | | | | | | |
| **Mono-objetivo** | **MSE ± SD** | 1.35E-2 ± 7.729E-3 | 2.20E-4 ± 1.147E-4 | 5.91E-5 ± 1.544E-5 | 1.80E-2 ± 3.736E-3 | 8.17E-2 ± 1.792E-2 | 1.49E+2 ± 1.331E+2 | 6.13E-12 ± 2.644E-12 |
| **Multi-objetivo** | **MSE ± SD** | 1.01E-1 ± 3.052E-2 | 1.62E-3 ± 3.908E-4 | 2.09E-4 ± 6.056E-5 | 7.00E-2 ± 5.995E-3 | 1.67E-1 ± 3.316E-2 | 1.31E+3 ± 3.661E+2 | 2.94E+0 ± 7.598E-1 |
| **BOSQUE ALEATORIO, CON 100 ÁRBOLES** | | | | | | | | |
| **Mono-objetivo** | **MSE ± SD** | 1.03E-2 ± 7.604E-3 | 1.67E-4 ± 8.348E-5 | 4.88E-5 ± 1.079E-5 | 1.64E-2 ± 3.830E-3 | 6.99E-2 ± 9.983E-3 | 9.47E+1 ± 1.198E+2 | 4.36E-12 ± 2.541E-12 |
| **Multi-objetivo** | **MSE ± SD** | 9.15E-2 ± 3.077E-2 | 1.49E-3 ± 4.168E-4 | 1.87E-4 ± 5.985E-5 | 6.64E-2 ± 6.774E-3 | 1.62E-1 ± 2.898E-2 | 1.23E+3 ± 4.642E+2 | 2.79E+0 ± 9.851E-1 |

Como se puede observar en la Tabla 7, los resultados en el conjunto datos *WAVEFORM* son muy positivos. El número de nodos para un árbol mono-objetivo es aproximadamente el mismo que para el árbol multi-objetivo, teniendo en cuenta que el árbol multi-objetivo predice las mismas variables que los 5 árboles mono-objetivo, y que el árbol muli-objetivo obtiene mejores resultados en 3 de esas variables, siendo la diferencia en el resto de variables bastante pequeña.

En el conjunto de datos *MUSIC*, como se puede observar en la Tabla 8, se aprecia que el aumento del error entre los modelos mono-objetivo y multi-objetivo es mucho mayor. No parece que haya cambiado la proporción en la que se reducen los nodos, que parece estar relacionada con el número de variables dependientes. Tampoco se aprecia una mejoría en la considerable diferencia de las tasas de errores cuando aumentamos el número de árboles en el bosque de 10 a 100.

# 5 Conclusiones y trabajo futuro

## 5.1 Conclusiones

En este trabajo, se ha planteado un estudio de modelo de regresión multi-objetivo basado en árboles CART y bosques aleatorios, introduciendo la perspectiva multi-objetivo en una biblioteca estadística ya probada y validada, y comparando las implementaciones clásicas con nuestras nuevas implementaciones. Después de validar la corrección del nuevo código, hemos hecho pruebas experimentales comparando los errores cuadráticos medios de los distintos modelos sobre 2 conjuntos de datos y sus varias variables dependientes.

Hemos comprobado que, aunque la disminución del número de nodos totales para predecir todas las variables dependientes siempre es mucho menor en el caso de los modelos multi-objetivo, sus tasas de error dependen mucho de la naturaleza del conjunto de datos escogido. En el conjunto de datos *WAVEFORM*, se ha encontrado que los modelos multi-objetivo, tanto el árbol CART como el bosque aleatorio, son en realidad ligeramente mejores que sus versiones clásicas.

Sin embargo, no podemos decir lo mismo de los modelos entrenados con el conjunto de datos *MUSIC*, que aunque mantenían las mejoras de eficiencia computacional relacionadas con la disminución del número de nodos totales, presentaban en sus versiones multi-objetivo una adición destacable a las tasas de error.

## 5.2 Trabajo futuro

En el contexto de las implementaciones realizadas dentro de este Trabajo de Fin de Grado **para árboles CART y bosques aleatorios**, proponemos implementar algoritmos de poda para los árboles de decisión, aleatorizar en la construcción de cada nodo el subconjunto de variables objetivo para el cual se quiere optimizar la división del nodo y además, estudiar de forma experimental temas que quedan fuera del alcance de este Trabajo de Fin de Grado, como:

- la coherencia que un enfoque multi-objetivo tiene en el rango de predicciones de estos clasificadores,
- si este enfoque permite explicitar las relaciones entre las variables dependientes,
- cuáles son las condiciones del conjunto de datos para que el enfoque multiobjetivo sea más exacto,

- si el enfoque multiobjetivo reduce la variabilidad en los errores de las predicciones a lo largo de las distintas variables dependientes,
- etc.

Otra posible avenida de trabajo futuro es la re-implementación y el estudio desde una perspectiva multi-objetivo de **otros algoritmos de regresión** tales como las SVM o las redes neuronales.

Por último, dada la empinada curva de aprendizaje a la hora de agregar nueva funcionalidad a la **biblioteca estadística** del Dr. Gonzalo Martínez, se proponen las siguientes acciones de ingeniería del software:

- Documentación dinámica propia de C++, realizada por ejemplo en Doxygen, para que en todo momento un nuevo desarrollador pueda conocer qué hace una función.
- Manual de usuario, para que un nuevo usuario de la biblioteca no tenga que consultar el código fuente.
- Encapsulado de las funciones propias de GNU/Linux u otras librerías externas, para poder ofrecer implementaciones alternativas con funciones Windows y así poder compilar en ambos entornos.
- Refactorización del código, que puede comprender establecer un estándar para los nombres de variables, que deberían dar más importancia a la legiblidad que a la brevedad y deberían estar escritos en inglés.

# Referencias


[1] Google Inc., «Machine learning: Google Trends,» [En línea]. Available: https://trends.google.es/trends/explore?q=%2Fm%2F01hyh_.

[2] IMB Inc., «CART Algorithm,» [En línea]. Available: ftp://ftp.boulder.ibm.com/software/analytics/spss/support/Stats/Docs/Statistics/Algorithms/14.0/TREE-CART.pdf.

[3] L. Breiman y A. Cutler, «Random Forests for Scientific Discovery,» [En línea]. Available: https://www.stat.berkeley.edu/~breiman/RandomForests/berkeley_files/frame.htm.

[4] L. Breiman, «Bagging Predictors,» *University of California at Berkeley*, 1994.

[5] L. Breiman, «Consistency for a simple model of random forests,» *University of California at Berkeley*, 2004.

[6] L. Breiman y A. Cutler, «Random Forests,» University of California at Berkeley, [En línea]. Available: https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm. [Último acceso: 27 Junio 2017].

[7] J. Struyf y S. Džeroski, «Constraint Based Induction of Multi-objective,» *F. Bonchi and J.-F. Boulicaut (Eds.): KDID 2005, LNCS,* nº 3933, pp. 222-233, 2006.

[8] Wolfram Mathworld, «Bessel's Correction,» [En línea]. Available: http://mathworld.wolfram.com/BesselsCorrection.html.

[9] «The GNU Awk User’s Guide,» [En línea]. Available: https://www.gnu.org/software/gawk/manual/gawk.html.

[10] «bc,» [En línea]. Available: https://www.gnu.org/software/bc/.

[11] M. G, «Functions: BASH Programming - Introduction HOW-TO,» 27 Julio 2000. [En línea]. Available: http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-8.html.

[12] G. M. Muñoz, «Clasificación Mediante Conjuntos,» *Universidad Autónoma de Madrid*, 2006.

[13] F. Sembolini, *Uso de técnicas de aprendizaje automático para predecir propiedades en simulaciones cosmológicas hidrodinámicas*, Universidad Autónoma de Madrid.

[14] Microsoft Inc., «C++ Language Reference,» 11 April 2016. [En línea]. Available: https://docs.microsoft.com/en-us/cpp/cpp/cpp-language-reference.

[15] H. Blockeel, L. De Raedt y J. Ramon, «Top-down induction of clustering trees,» *Katholieke Universiteit Leuven*, 2000.

[16] L. Breiman, Classification and regression trees, 1984.

# Glosario

En esta sección presentamos definimos los tecnicismos y términos específicos utilizados en el presente documento.

| | |
|---|---|
| **Árbol CART** | Árbol de decisión con unas condiciones específicas creado por Leo Breiman. |
| **Bosque aleatorio** | Clasificador que su vez es un conjunto de clasificadores, en concreto, de árboles de decisión. |
| **Función de interfaz** | Función que la biblioteca estadística del Dr. Gonzalo Martínez expone a los usuarios de los *scripts*. |
| **MORT** | Siglas para *multi-objective regression tree*. Introducido en un estudio ya existente sobre árboles de decisión multi-objetivo. |
| **MSE** | Error cuadrático medio. |
| **Regresión clásica**<br>**Regresión mono-objetivo** | Regresión asumiendo que sólo hay 1 variable dependiente, y que el resto son independientes. |
| **Regresión multi-objetivo** | Regresión asumiendo que hay más de 1 variable dependiente. |
| **SVM** | Máquinas de vectores de soporte |
| **Validación cruzada** | Sistema de entrenamiento y validación de clasificadores, que genera parejas de subconjuntos de entrenamiento y *test*. |
| **Variable dependiente**<br>**Objetivo** | Atributo o atributos cuyo valor queremos predecir a partir de los valores del resto de variables. Variable cuyo valor asumimos que depende de los valores de otras variables. |
| **Variable independiente** | Atributo con cuyo valor predecimos el valor de las variables dependientes. Asumimos que su valor no depende de ninguna otra variable. |

# Anexos

## A Nueva funcionalidad del código

En este anexo detallamos todos los cambios y añadidos que se han acometido en la biblioteca del Dr. Gonzalo Martínez. Con el objetivo de presentar la información de la manera más ordenada posible, primero listamos en la estructura de carpetas los archivos involucrados en la Figura 2.

- **<raíz>**
  - **FES**
    - **Extracto**
      - data20.cpp
      - data20.h
  - **nucleo**
    - Arcing.cpp
    - Classifier.h
    - Ensemble.cpp
    - Ensemble.h
    - Evaluations.cpp
    - Evaluations.h
    - FnsClasificacion.cpp
    - FnsClasificacion.h
    - FnsClasificacion_v2.cpp
    - GPTree.cpp
    - GPTree.h
    - Language.cpp
  - generateGraphs.py
  - plotForestVariableSize.gpi
  - <scripts con extensión .mKo>

..... ⇒ Archivo modificado

_____ ⇒ Nuevo archivo

**Figura 2 Listado de código modificado**

A continuación, navegaremos paso por paso por cada pareja de archivos CPP/H, identificando las clases C++ nuevas o modificadas.

### A.1 data20

## Instance

*Instancia del conjunto de datos*

Se añade a esta y a todas sus subclases los métodos:

- `void` **`ChangeDependentColumn`**`(int newDependenColumnIndex)`
  Intercambia los valores de lo que se considera la variable dependiente, es decir, la última variable de la fila, con la variable señalada en el índice.
- `void` **`CopyDependentColumn`**`()`
  Considerando los valores de una instancia como una lista ordenada, inserta al final de esa lista un nuevo valor, igual al último valor de la lista original, que corresponde a lo que asumimos que es la variable dependiente.

Como estos métodos se han definidos como funciones virtuales puras en la clase *Instance*, es un requerimiento que aquellas de sus subclases que vayan a ser instaciadas definan una implementación [14]. Sin embargo, la mayoría de las implementaciones se limitan a lanzar una excepción de tipo `std::logic_error` avisando de que todavía no han sido terminadas. Las únicas dos implementaciones terminadas son las de las clases *BasicInstance* y *MultiobjectiveInstance*.

Además, se modifica el método estático **`NewInstanceByName`** para que tenga en cuenta la nueva subclase *MultiobjectiveInstance*.

## BasicInstance

*Instancia básica del conjunto de datos*

Se implementan **`ChangeDependentColumn`** y **`CopyDependentColumn`** teniendo en cuenta el *array* de valores *double* de la clase.

## MultiobjectiveInstance

*Instancia de un conjunto de datos con varias variables dependientes*

La principal motivación de esta nueva clase, que hereda de *BasicInstance*, es aportar una interfaz para consultar de manera estática el número de objetivos del conjunto de datos, y

para cada instancia, los valores de esos objetivos. Además, implemena **`ChangeDependentColumn`** y **`CopyDependentColumn`** para que se mantenga la coherencia de esa interfaz.

## MultipleMSECriterium

*Criterio de calificación por error cuadrático medio para varios objetivos*

Esta nueva clase, subclase de *SplitCriterium* y fuertemente basada en la clase *MSECriterium*, adapta el criterio de error cuadrático medio para ser usado en conjunto de datos con varios objetivos. Este criterio se usará tanto para la construcción de árboles CART como de bosques aleatorios.

Podemos encontrar una definición formal del concepto matemático subyacente en la ecuación del apartado Árboles multi-objetivo.

## Data

*Conjunto de datos*

Esta es la clase que representa los datos leídos desde un fichero, y que guarda referencias a las instancias y a propiedades como el número de variables. Se añaden a ella los métodos:

- `void` **`ChangeDependentColumn`**`(int newDependenColumnIndex)`
- `void` **`CopyDependentColumn`**`(int new_num_multi = -1)`

Estos métodos, además de actualizar de manera interna los campos de la clase, llaman de manera iterativa a los métodos del mismo nombre de las instancias que componen el conjunto de datos.

Además, también se han modificado los métodos **`SetDefaultNames`** y **`GetNames`** para que, en lugar de asumir que el número de variables independientes es el número de variables menos uno, se tenga también en cuenta el número de objetivos.

## NomData

*Implementación de conjunto de datos*

Se ha modificado el método **`WriteNode`** de esta clase para que, en el caso de imprimir por pantalla un árbol de decisión, pinte en los nodos hoja las predicciones para todas las variables objetivo, no sólo para la última variable.

## A.2 Arcing

# ArcingTemplate

El único cambio que se ha hecho en esta clase, que está involucrada en la construcción de conjuntos de clasificadores, es comentar la llamada a la función **`EndArc`**, que estaba entorpeciendo nuestro desarrollo y que, para los modelos estudiados en este Trabajo de Fin de Grado, era irrelevante.

## A.3 Classifier

# Classifier

*Modelo clasificador*

Se añaden a la clase los métodos:

- `double` **`Average`**`(int ElementIndex, int AttributeIndex)`
  Este método es en realidad una sobrecarga de un método existente, pero permitiendo especificar el índice del atributo cuya predicción se quiere obtener. Esto nos será útil cuando deseemos predecir un objetivo determinado en un clasificador multi-objetivo. Sin embargo, la implementación de este método en la clase *Classifier* se limita a lanzar una excepción avisando de que el método no está terminado.
- `std::vector<double>` **`MultipleAverage`**`(int ElementIndex)`
  Devuelve la lista de predicciones para todas las variables objetivo. Al igual que la nueva sobrecarga del método **`Average`**, la implementación en esta clase sólo lanza una excepción.

## A.4 Ensemble

# Ensemble

*Conjunto de clasificadores*

Además de añadir referencias en la declaración de la clase a los métodos de la clase padre *Classifier* cuyos nombres son **`Average`** y **`MultipleAverage`**, se añade a la clase el método:

- `void` **`RegressionError`**`(int first, int last, std::vector<double> *errors, int dVar=-1)`
  Establece una lista de tasas de error por cada instancia del conjunto de datos.

### A.5 Evaluations

## MultipleMSEEvaluation

*Evaluación de clasificador por error cuadrático medio para varios objetivos*

Esta nueva clase, que hereda de la clase *Evaluation*, se basa fuertemente en la clase *MSEEvaluation*, de manera análoga a como la clase *MultipleMSECriterium* se basa en la clase *MSECriterium*. Para evaluar el clasificador, utiliza el nuevo método **`MultipleAverage`** de la clase *Classifier*.

## MSEEvaluation

*Evaluación de clasificador por error cuadrático medio*

Se actualiza el método **`MSEEvaluation`** para que pueda recibir un nuevo parámetro opcional que defina el índice de la variable objetivo con la cual el usuario desea evaluar el clasificador. Este nuevo modo utiliza la sobrecarga de la función **`Average`** definida en la clase *Classifier*.

## Evaluation

*Evaluación de clasificador*

Se actualiza el método **`EvaluationByClassName`** para que tenga en cuenta la clase *MultipleMSEEvaluation*.

### A.6 FnsClasificacion

## LoadDataSet

*Función de interfaz para cargar un conjunto de datos*

Se modifica la implementación de esta clase para permitir, de manera opcional, asignar a la ejecución un número de variables objetivo mayor que 1. Esta cantidad se almacena de manera estática en la clase *MultiobjectiveInstance*.

## ChangeDependentColumn

*Función de interfaz para cambiar la columna dependiente*

Esta nueva clase desvela la funcionalidad definida en el método **`ChangeDependentColumn`** de la clase *Data* a los usuarios de los *scripts* como una nueva función de interfaz.

## CopyDependentColumn

*Función de interfaz para duplicar la columna dependiente*

Esta nueva clase desvela la funcionalidad definida en el método **`CopyDependentColumn`** de la clase *Data* a los usuarios de los *scripts* como una nueva función de interfaz.

## RegressionError

*Función de interfaz para duplicar la columna dependiente*

Esta nueva clase desvela la funcionalidad definida en el método **`RegressionError`** de la clase *Ensemble* a los usuarios de los *scripts* como una nueva función de interfaz, permitiendo además guardar la lista de tasas de error generadas en un nuevo fichero.

### A.7 GPTree

## DecisionTree

*Árbol de decisión*

En esta clase, que hereda de *Classifier*, implementamos la nueva sobrecarga de **`Average`** y el nuevo método **`MultipleAverage`** definidos en la clase *Classifier*, de manera que se consulten correctamente las predicciones en la estructura de nodos interna.

### A.8 Language

## StringRepositoryEnglish y StringRepositorySpanish

*Nombres de funciones de interfaz en inglés y castellano, respectivamente*

Se añaden a los repositorios de nombres de funciones de interfaces las funciones definidas en el apartado FnsClasificacion.

### A.9 Scripts

## generateGraphs.py

Este script escrito en Python se puede considerar como el script maestro, desde donde se llama al resto de scripts para generar las particiones de validación cruzada, evaluar los modelos de datos, y pintar las gráficas (aunque no se han incluido en este memoria, se ha desarrollado un script GNUPLOT para pintar gráficas en las que se estudia el comporamiento del MSE frente al tamaño del bosque aleatorio).